

GRASP — General Responsibility Assignment Software Patterns Object Oriented Analysis and Design

Aron Trauring

T++ Technical Skills Training Program

CUNY Institute for Software Design & Development (CISDD)

New York Software Industry Association (NYSIA)

December 20th, 2004

Introduction to GRASP Exercise

Use Case Realization

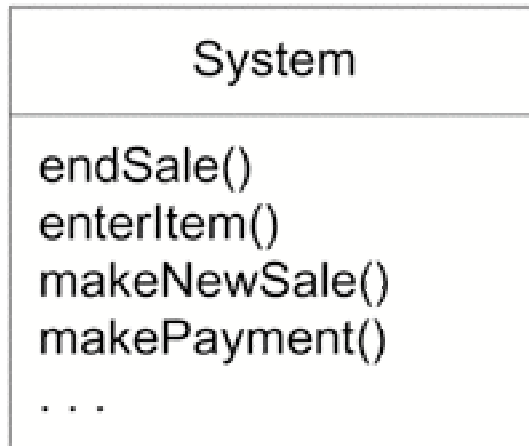
- How a particular use case is realized within the Design Model in terms of collaborating objects
- Use case suggests the system operations
- System operations becoming starting messages entering the controllers for domain layer interaction diagrams
- Domain layer interaction diagrams illustrate the use case realization
- Domain model “inspires” classes but not one-to-one
- Use postconditions from contract to design detailed message interactions

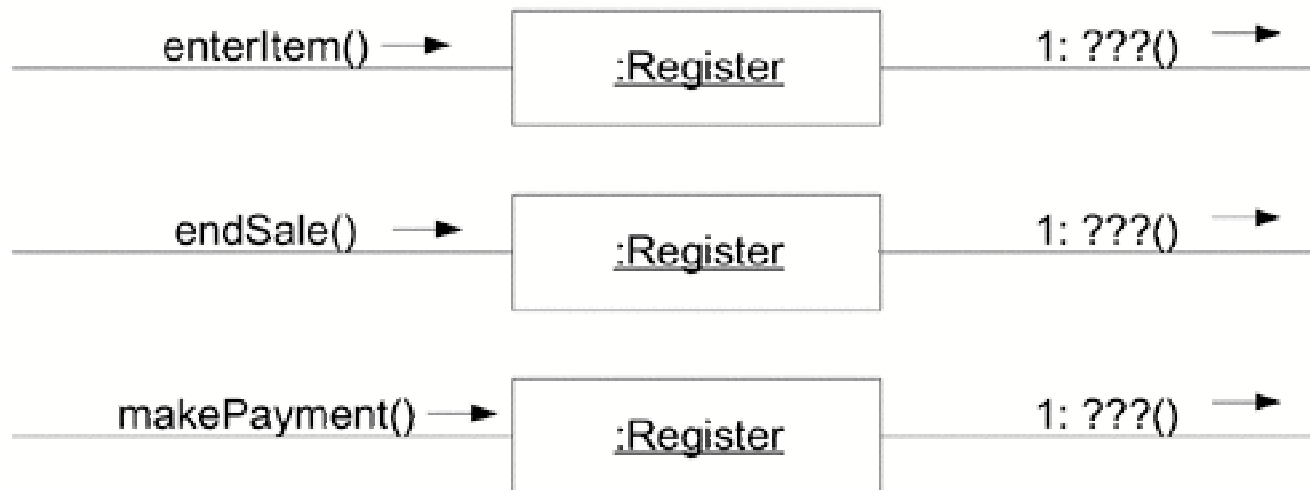
T++ — CISDD — NYSIA

GRASP - OOAD

- Customer needs to be involved in the process throughout

System Operations and Interaction Diagrams — Example





System Operations and Contracts — Example

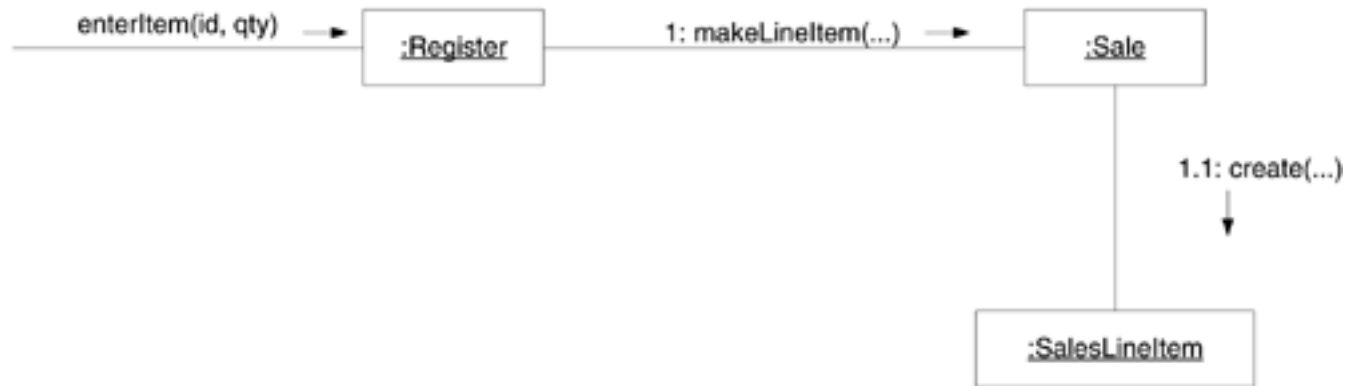
Contract CO2: enterItem

Operation: EnterItem(itemID : ItemID, quantity : integer)

Cross References: Use Cases: Process Sale

Preconditions: There is a sale underway.

Postconditions: - A SalesLineItem instance sli was created (instance creation)



Guidelines to Keep in Mind

- Model Start-Up after you've modeled other stuff and know what you need to initialize
- Keep things that need to be initialized and connected at start-up (e.g. to persistence classes)
- Start assigning responsibilities by clearly stating responsibilities
- Use GRASP principles

Visibility

- Visibility — the ability of one object to “see” or have a reference to another object
- Visibility is required for one object to message another
- *Attribute Visibility* — Y is an attribute of ClassX
- *Parameter Visibility* — Y is a parameter of MethodX
- *Local Visibility* — Y is a (non-parameter) local object in a method of ClassX
- *Global Visibility* — Y is globally visible (language dependent)

Polymorphism

Polymorphism

Problem: How to avoid case logic?

Solution: Use polymorphism

Example: `getTaxes(Sale)`, `landedOn(Square)`

Pure Fabrication

Pure Fabrication

Problem: What object to assign responsibility when assigning to Domain Concept objects will violate High Cohesion or Low Coupling or some other goal.

Solution: Create a made up class (not in domain model) which supports High Cohesion

Example: PersistentStorage

Indirection

Indirection

Problem: How to de-couple objects to support low-coupling and increase reuse potential?

Solution: Assign responsibility to an intermediate object to mediate between other components and services

Example: TaxMasterAdaptor (between Sale and external tax systems)

Protected Variation

Protected Variation

Problem: How to design objects so that variation or instability does not have an undesirable impact on other elements?

Solution: Identify points of predicated variation or instability; assign responsibilities to create a stable interface around them

Example: TaxMasterAdaptor or look-up services (locations may vary)

GRASP and GOF

- GRASP patterns can be seen as generalization of GRASP patterns
- Example:

Adaptor

Problem: How to resolve incompatible interfaces or provide a stable interface to stable components with different interfaces?

Solution: Convert the original interface of the component into another interface, through an intermediate adaptor object

Example: TaxCalculatorAdaptor

GRASP: Protected Variation from third-party interfaces by using Indirection and Polymorphism